Spread Operator

The main objective of the spread operator is to **spread** the elements of an array or object. This is best explained with examples.

# Apply

A common use case is to spread an array into the function arguments. Previously you would need to use Function.prototype.apply:

function foo(x, y, z) { }

var args = [0, 1, 2];

foo.apply(null, args);

Now you can do this simply by prefixing the arguments with ... as shown below:

function foo(x, y, z) { }

var args = [0, 1, 2];

foo(...args);

Here we are **spreading** the args array into positional arguments.

# Destructuring

We've already seen one usage of this in destructuring:

var [x, y, ...remaining] = [1, 2, 3, 4];

console.log(x, y, remaining); // 1,2,[3,4]

The motivation here is to simply make it easy for you to capture the remaining elements of an array when destructuring.

# Array Assignment

The spread operator allows you to easily place an expanded version of an array into another array. This is demonstrated in the example below:

var list = [1, 2];

list = [...list, 3, 4];

console.log(list); // [1,2,3,4]

You can put the expanded array in at any position, and get the effect you'd expect:

var list = [1, 2];

list = [0, ...list, 4];

console.log(list); // [0,1,2,4]

# Object spread

You can also spread an object into another object. A common use case is to simply add a property to an object without mutating the original:

const point2D = {x: 1, y: 2};

/\*\* Create a new object by using all the point2D props along with z \*/

const point3D = {...point2D, z: 3};

For objects, the order of where you put the spread matters. This works something like Object.assign, and does what you'd expect: what comes first is 'overridden' by what comes later:

const point2D = {x: 1, y: 2};

const anotherPoint3D = {x: 5, z: 4, ...point2D};

console.log(anotherPoint3D); // {x: 1, y: 2, z: 4}

const yetAnotherPoint3D = {...point2D, x: 5, z: 4}

console.log(yetAnotherPoint3D); // {x: 5, y: 2, z: 4}

Another common use case is a simple shallow extend:

const foo = {a: 1, b: 2, c: 0};

const bar = {c: 1, d: 2};

/\*\* Merge foo and bar \*/

const fooBar = {...foo, ...bar};

// fooBar is now {a: 1, b: 2, c: 1, d: 2}

# Summary

apply is something that you often use in JavaScript, so it's good to have a better syntax where you don't have that ugly null for the this argument. Also having a dedicated syntax for moving arrays out of (destructuring) or into (assignment) other arrays provides a neat syntax for when you are doing array processing on partial arrays.